FICHEROS BINARIOS

Los **ficheros binarios** son aquellos que **almacenan la información en formato binario (bits)**, tal como la entiende la máquina (no como texto legible por humanos).

A diferencia de los ficheros de texto:

- No usan saltos de línea ni separadores.
- Los datos se guardan tal cual están en memoria.
- Son más **eficientes y compactos**.
- **No se pueden leer directamente** en un editor de texto.

Tamaño de los tipos de datos en binario

Tipo de dato	Tamaño (bytes)	Descripción
`byte`	1	Número entero pequeño (-128 a 127)
`short`	2	Entero corto (-32.768 a 32.767)
`int`	4	Entero estándar
`long`	8	Entero largo
`float`	4	Número real simple precisión
`double`	8	Número real doble precisión
`char`	2	Carácter Unicode
`boolean`	1	Verdadero o falso (internamente 0 o 1 aprox.)

Por ejemplo:
Un registro con:

- int id → 4 bytes
- double salario → 8 bytes
- char[10] nombre → 20 bytes

Tendría un total de **32 bytes por registro.**

Clases principales para trabajar con binarios en Java

FileOutputStream

Sirve para **escribir bytes** en un fichero binario.

```
import java.io.FileOutputStream;
import java.io.IOException;

public class EjemploFileOutputStream {
    public static void main(String[] args) {
        String texto = "Hola binario";
        try (FileOutputStream fos = new FileOutputStream("ejemplo.dat")) {
            fos.write(texto.getBytes()); // convierte texto a bytes y los escribe
            System.out.println("Fichero binario creado con FileOutputStream");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OJO: Usa getBytes() porque FileOutputStream trabaja a **nivel de bytes**, no de caracteres.

FileInputStream

Sirve para **leer bytes** desde un fichero binario.

```
import java.io.FileInputStream;
import java.io.IOException;

public class EjemploFileInputStream {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("ejemplo.dat")) {
            int byteLeido;
            while ((byteLeido = fis.read()) != -1) {
                System.out.print((char) byteLeido); // Convertimos a char para ver el texto
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OJO: fis.read() devuelve un int (de 0 a 255). Cuando devuelve -1, significa **fin del fichero**.

RandomAccessFile

Permite **leer y escribir** datos binarios o texto desde la posición del puntero, en cualquier parte del fichero.

Clase	Propósito	Tipo de acceso	Ejemplo de uso
`FileInputStream`	Leer bytes de un fichero	Secuencial	Leer imagen o fichero `.dat`
`FileOutputStream`	Escribir bytes en un fichero	Secuencial	Guardar datos en binario
`RandomAccessFile`	Leer/escribir datos primitivos	Aleatorio (mover puntero)	Crear fichero binario de registros

Modos de apertura

- "r": solo lectura.
- "rw": lectura y escritura (crea el archivo si no existe).

Control del puntero y tamaño

- getFilePointer() → posición actual (long).
- seek(long posicion) → mueve el puntero.
- length() → tamaño actual del archivo.
- setLength(long newLen) → **trunca o extiende** el archivo (si extiendes, los bytes nuevos quedan en cero).

Lectura / escritura disponibles

Lee y escribe **tipos primitivos** y algunas variantes:

- Lectura: read(), read(byte[]), readBoolean(), readByte(), readChar(), readShort(), readInt(), readLong(), readFloat(), readDouble(), readFully, readUTF().
- Escritura: write(int), write(byte[]), writeBoolean(), writeByte(), writeChar(), writeUTF().

Fijar formato del registro de cadenas de caracteres

Para que los registros de las cadenas de texto tengan una longitud fija, debemos fijarla.

Veremos dos métodos: .setlenght() aplicado a un buffer y String.format

.setlenght():

setLength(0) → **vacía** el archivo.

setLength(n) menor → **trunca**.

setLength(n) mayor → **extiende** (los bytes nuevos quedan como ceros).

Ejemplo:

```
StringBuffer sb = new StringBuffer(nombre);
sb.setLength(10);            // recorta o rellena con '\u0000'
raf.writeChars(sb.toString()); // escribe 10 chars (20 bytes)
```

Lectura correspondiente:

```
char[] nombre = new char[10];
for (int i = 0; i < 10; i++) nombre[i] = raf.readChar();
String nombreStr = new String(nombre).trim(); // quita ' ' y '\u0000'
```

String.format:

La forma general sería:

String.format (%[flags][width][.precision]conversión)

Donde:

% → empieza el formato.

flags (opcionales):

- - → alinear a la **izquierda** (por defecto es derecha).
- 0 → para números, rellena con **ceros** (conversión numérica).

width → **anchura mínima** (en caracteres). Si el texto es más corto, se **rellena** (con espacios salvo que se indique otra cosa).

.precision (opcional) → Con **%s** (strings): **máximo** de caracteres a mostrar (recorta si hay más).

conversion → tipo: s (String), d (entero), f (float/double), etc.

Ejemplos aplicados:

String fijo = String.format("%-" + len + "." + len + "s", s);

% → comienzo.

- → alinear a la izquierda.

len (width) → mínimo len caracteres (si falta, rellena con espacios a la derecha).

. + len (precision) → máximo len caracteres (si sobra, recorta).

s → trata el argumento como String.

Con len = 6:

"Ana" → "Ana " (3 letras + 3 espacios).

"Martha" → "Martha" (ya mide 6, ni recorta ni rellena).

"Demasiado" → "Demasi" (recorta a 6).

Otros ejemplos rápidos

String.format("%6s", "Ana"); // "   Ana" (derecha, rellena a la izquierda)

String.format("%-6s", "Ana"); // "Ana   " (izquierda, rellena a la derecha)

String.format("%06d", 42);   // "000042" (número, relleno con ceros)

Escritura con RandomAccessFile (crear Empleados.dat):

```java
import java.io.IOException;
import java.io.RandomAccessFile;

public class EmpleadosBinarios {
    public static void main(String[] args) {
        String[] nombres = {"Ana", "Luis", "Marta", "Carlos", "Eva"};
        int[] dept = {10, 20, 10, 30, 10};
        double[] salario = {1500.0, 1800.5, 1600.0, 2100.75, 1700.25};

        try (RandomAccessFile raf = new RandomAccessFile("Empleados.dat", "rw")) {
            for (int i = 0; i < nombres.length; i++) {
                raf.writeInt(i + 1); // ID
                StringBuffer sb = new StringBuffer(nombres[i]);
                sb.setLength(10); // 10 caracteres (20 bytes)
                raf.writeChars(sb.toString());
                raf.writeInt(dept[i]); // departamento
                raf.writeDouble(salario[i]); // salario
            }

            System.out.println("Fichero binario creado correctamente.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

¿Qué pasa en el bucle for?

Por cada empleado se escribe un **registro** con este formato fijo (36 bytes):

- **ID**: writeInt(i + 1) → 4 bytes.
- **Apellido**:
  - StringBuffer sb = new StringBuffer(nombres[i]) crea una cadena mutable.
  - sb.setLength(10) garantiza **exactamente 10 caracteres** (si falta, rellena con espacios; si sobra, recorta).
  - writeChars(...) escribe **carácter a carácter** en **Unicode** (2 bytes por char) → **20 bytes**.
- **Departamento**: writeInt(...) → 4 bytes.
- **Salario**: writeDouble(...) → 8 bytes (IEEE-754).

- **Total por registro**: 4 + 20 + 4 + 8 = **36 bytes**.

Tras escribir un registro, el puntero avanza automáticamente 36 bytes; al terminar, el fichero ocupa nEmpleados × 36.

Al usar longitud fija en el apellido (10 chars), cada registro pesa lo mismo. Eso permite "saltar" a cualquier registro con seek() (acceso aleatorio real).

Lectura con RandomAccessFile (mostrar Empleados.dat)

```
public class LeerEmpleadosBinarios {
    public static void main(String[] args) {
        try (RandomAccessFile raf = new RandomAccessFile("Empleados.dat", "r")) {
            int id, dept;
            double salario;
            char[] nombre = new char[10];
            String nombreStr;

            while (raf.getFilePointer() < raf.length()) {
                id = raf.readInt();
                for (int i = 0; i < nombre.length; i++) {
                    nombre[i] = raf.readChar();
                }
                nombreStr = new String(nombre).trim();
                dept = raf.readInt();
                salario = raf.readDouble();

                System.out.printf("ID: %d | Nombre: %-10s | Dpto: %d | Salario: %.2f%n",
                        id, nombreStr, dept, salario);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

- **Condición del while**:
    - raf.getFilePointer() es la **posición actual** del puntero (en bytes).
    - raf.length() es el **tamaño total** del fichero.
    - Mientras el puntero no llegue al final, queda **al menos un registro** por leer.
- **Lecturas** en el **mismo orden** en que se escribieron:

- readInt() → recupera el **ID** (4 bytes).

- 10 veces readChar() → reconstruye el **apellido** (20 bytes); luego trim() elimina espacios añadidos al fijar longitud.

- readInt() → **departamento** (4 bytes).
- readDouble() → **salario** (8 bytes).

El puntero avanza solo; al acabar un registro, queda colocado al inicio del siguiente.

OJO: el **orden de lectura** debe ser **exactamente el mismo** que el de escritura y con las mismas longitudes. Si cambias el orden o las longitudes, "descuadras" el puntero y leerás basura.

EJERCICIOS EJEMPLO:

1) Escribir y leer enteros secuencialmente

```java
import java.io.RandomAccessFile;
import java.io.IOException;

public class Ej1 {
    public static void main(String[] args) throws IOException {
        // Abre (o crea) un archivo binario con permisos de lectura y escritura.
        try (RandomAccessFile raf = new RandomAccessFile( pathname: "nums.bin",  mode: "rw")) {
            // Borra el contenido previo para que la prueba sea limpia.
            raf.setLength(0);

            // Escribe 5 enteros cualquiera: 10, 20, 30, 40, 50 (cada int ocupa 4 bytes).
            for (int i = 1; i <= 5; i++) {
                raf.writeInt( v: i * 10); // escribe y avanza el puntero 4 bytes
            }

            // Vuelve el puntero al inicio del archivo (byte 0).
            raf.seek( pos: 0);

            // Lee 5 enteros desde el comienzo y los imprime.
            for (int i = 0; i < 5; i++) {
                int valor = raf.readInt(); // lee 4 bytes como int y avanza
                System.out.println(valor); // muestra el valor leído
            }
        } // try-with-resources cierra el archivo automáticamente
    }
}
```

2) Acceso directo a una posición fija

```java
import java.io.RandomAccessFile;
import java.io.IOException;

public class Ej2 {
    public static void main(String[] args) throws IOException {
        // Cada 'int' son 4 bytes; tendremos 3 "slots": 0, 4 y 8.
        try (RandomAccessFile raf = new RandomAccessFile( pathname: "slots.bin",  mode: "rw")) {
            raf.setLength(0); // limpia el archivo

            int[] datos = {100, 200, 300}; // valores a guardar
            for (int i = 0; i < datos.length; i++) {
                long offset = i * 4L;     // calcula posición: slot i * 4 bytes
                raf.seek(offset);         // mueve el puntero a ese byte
                raf.writeInt(datos[i]);   // escribe el entero en esa posición
            }

            // Queremos leer solo el segundo elemento (i=1 → byte 4).
            raf.seek( pos: 1 * 4L);              // posiciona el puntero en el byte 4
            int segundo = raf.readInt();  // lee 4 bytes (200)
            System.out.println("Segundo = " + segundo);
        }
    }
}
```

3) Cadenas cortas con writeUTF/readUTF

```java
import java.io.RandomAccessFile;
import java.io.IOException;

public class Ej3 {
    public static void main(String[] args) throws IOException {
        // writeUTF guarda longitud + bytes (formato Modified UTF-8).
        try (RandomAccessFile raf = new RandomAccessFile( pathname: "nombres.bin",  mode: "rw")) {
            raf.setLength(0);             // limpia
            raf.writeUTF( str: "Ana");          // escribe "Ana" y su longitud
            raf.writeUTF( str: "Luis");         // escribe "Luis"
            raf.writeUTF( str: "Marta");        // escribe "Marta"

            raf.seek( pos: 0);                  // vuelve al inicio para leer
            System.out.println(raf.readUTF()); // lee 1ª cadena tal como se escribió
            System.out.println(raf.readUTF()); // lee 2ª cadena
            System.out.println(raf.readUTF()); // lee 3ª cadena
        }
    }
}
```

4) Registros de tamaño fijo (id + nombre20 + edad)

```java
import java.io.RandomAccessFile;
import java.io.IOException;

public class Ej4 {
    // Número de caracteres fijos para el nombre
    static final int NCHARS = 20;  no usages
    // Tamaño de un registro en bytes: id(4) + nombre(20 chars * 2B) + edad(4) = 48
    static final int BYTES_REG = 4 + 2 * NCHARS + 4;  no usages

    public static void main(String[] args) throws IOException {
        try (RandomAccessFile raf = new RandomAccessFile( pathname: "personas.dat",  mode: "rw")) {
            raf.setLength(0); // limpia el archivo

            // ----- Escribir registro 0 -----
            raf.seek( pos: 0 * BYTES_REG);      // salta al inicio del registro 0
            raf.writeInt( v: 1);               // id = 1
            escribirNombreFijo(raf,  s: "Ana", NCHARS); // escribe 20 chars
            raf.writeInt( v: 30);              // edad = 30

            // ----- Escribir registro 1 -----
            raf.seek( pos: 1 * BYTES_REG);      // salta al registro 1
            raf.writeInt( v: 2);               // id = 2
            escribirNombreFijo(raf,  s: "Luis", NCHARS);
            raf.writeInt( v: 25);              // edad = 25

            // ----- Leer registro 1 directamente -----
            raf.seek( pos: 1 * BYTES_REG);      // coloca el puntero al inicio del reg. 1
            int id = raf.readInt();        // lee id (4 bytes)
            String nombre = leerNombreFijo(raf, NCHARS); // lee 20 chars y recorta
            int edad = raf.readInt();      // lee edad (4 bytes)

            System.out.println(id + " " + nombre + " " + edad);
        }
    }

    // Escribe exactamente 'len' caracteres: recorta o rellena con espacios.
    static void escribirNombreFijo(RandomAccessFile raf, String s, int len) throws IOException {  no usages
        s = (s == null) ? "" : s;                // evita null
        if (s.length() > len) s = s.substring(0, len); // recorta si sobra
        s = String.format("%-" + len + "s", s);        // pad con espacios a la derecha
        for (int i = 0; i < len; i++) {
            raf.writeChar(s.charAt(i));          // cada char = 2 bytes (UTF-16BE)
        }
    }

    // Lee exactamente 'len' chars y hace trim() para quitar espacios de relleno.
    static String leerNombreFijo(RandomAccessFile raf, int len) throws IOException {  no usages
        StringBuilder sb = new StringBuilder(len); // buffer de chars
        for (int i = 0; i < len; i++) {
            sb.append(raf.readChar());           // lee 2 bytes como char
        }
        return sb.toString().trim();             // recorta los espacios
    }
}
```